

---

# VR Juggler

FAQ

\$Date: 2002/02/20 00:55:47 \$

## General Information

- 1.1. Is there a FAQ for VR Juggler?

Yes, check out <http://www.vrjuggler.org/> where you will find a FAQ on the menu to the left. Submit any new questions, or any solutions you've found that are useful to the user mail list [[http://www.sourceforge.net/mail/?group\\_id=8041](http://www.sourceforge.net/mail/?group_id=8041)].

- 1.2. Are there any applications (other than the demos) that can be downloaded?

<mailto:vrjuggler-contact@vrjuggler.org>Not yet, but we will take submissions! You could become famous with your cool app! Mail it to us.

- 1.3. What is VR Juggler?

VR Juggler is a C++ class library that is used as a framework for application development. VR Juggler controls the low-level aspects of the system on which the user application is executing. It provides a generic interface to common input and output devices that VR Juggler can control. VR Juggler controls the following:

- Graphics rendering
- VR devices
- Graphic projection setup
- Multi-processing
- Process synchronization

- 1.4. What is VR Juggler not?

VR Juggler does not have its own graphics API. It requires you to use a standard API such as OpenGL. VR Juggler does not have any support for scene graphs or other data structures for the application graphics. VR Juggler is not a visual development environment.

- 1.5. Great, where do I begin?

To begin, we suggest reading one of VR Juggler's books called the *Getting Started Guide*. You will find this book in the tree control on the VR Juggler website [<http://www.vrjuggler.org/>]. In the tree control, the path to getting started is: "Documentation"/"Getting Started".

- 1.6. What do I need to run a VR Juggler application?

For a very specific answer to this question, check out the downloads page [<http://www.vrjuggler.org/html/download/>]. There will be requirements next to each VR Juggler distribution. Generally you will need OpenGL or OpenGL Performer as your image generator. VR Juggler will run by itself otherwise (you will not need any other display driver such as CAVElib™, or World Toolkit™).

1.7. What C++ compilers are supported?

The following lists supported platforms and the compilers that we use in developing and distributing versions of VR Juggler for those platforms: If you do not have the supported compiler for your operating system, we will be limited in how much we can help you.

- *IRIX*: we require the MIPSpro Compilers, version 7.3.1.1m or higher and the associated STL implementation. Previous versions had bugs that caused VR Juggler to crash.
- *Win32*: we only support Microsoft Visual C++ 6.0 with Service Pack 4. You can use Service Pack 3, but it is recommended that you upgrade if possible. There have been reports of VR Juggler compiling with Visual C++ 5.0, but we do not officially support that version.
- *Other platforms*: we support the latest versions of GCC only. You must use at least GCC 2.95.2. GCC 3.0 is also supported.

1.8. How do I compile the VR Juggler source code under Win32?

You really don't need to do this unless you plan to modify the code and submit the changes to us. You can find precompiled binaries for Win32 on our downloads page [<http://www.vrjuggler.org/html/download/>]. To compile the source code to VR Juggler on Win32 platforms, you will want to download the Cygwin set of UNIX tools [<http://sources.redhat.com/cygwin/download.html>] including Perl 5.6.1, GNU Autoconf, and GNU Automake. You can find detailed information about all of this in “Documentation”/“Juggler Team Guide”/“Build Systems”/“Juggler”/“Win32 Caveats” of the *Juggler Team Guide* (book) on our web site.

## VjControl

2.1. Why doesn't VjControl run?

Prior to the release of VR Juggler 0.1.97, VjControl would work with Java 1.1 provided that the Java Swing classes were available in a JAR file called `swingall.jar`. VjControl purposely ignores your `$CLASSPATH` environment variable, so with the old releases we recommend that you place `swingall.jar` file into your `bin` directory along with the `vjcontrol` or `vjcontrol1.1` executable. Since VR Juggler 0.1.97, VjControl *only* works with Java 2 or newer (JDK versions 1.2 and higher). Because of this, `swingall.jar` is no longer needed.

Another possibility is that you do not have the `$VJ_BASE_DIR` environment variable set. Setting this variable is *required* to run VjControl because the shell script (or batch file on Win32) needs to be able to find the file `VjControl.jar` which is always located in `$VJ_BASE_DIR/bin` with the `vjcontrol` script and `vjcontrol.bat` batch file. Refer to the documentation describing environment variables [<http://www.vrjuggler.org/html/getting.started/environment.variables.html>] for more information. It explains all the environment variables related to VR Juggler (and VjControl).

2.2. What is `swingall.jar`?

The file `swingall.jar` contains the Swing classes that are part of the Java 2 distribution. These classes were need to use Swing with Java 1.1. You may need to install the Java Swing distribution from Sun [<http://www.sun.com/>]. After you do that, find the file `swingall.jar` (in the Java installed location), and copy it to the same place as where the **vjcontrol** executable is (usually in `$VJ_BASE_DIR/bin`).

## VR Applications

### 3.1.

When I run on IRIX, the application hangs right after printing “New application set”

This most often happens when using the IRIX SPROC version of VR Juggler and linking the POSIX thread library (`libpthread`) into the application. When you compile your VR Juggler application, make sure not to use `-lpthread`. If your application requires POSIX threads, use the pthreads version of VR Juggler instead.

On IRIX, there are two ways to write multithreaded software, and VR Juggler supports both. The methods are IRIX SPROC and POSIX threads. These two systems are incompatible, and a SPROC application will hang the first time it tries to create a thread if the POSIX thread library is linked. Removing `-lpthread` fixes the problem.

### 3.2.

Why is there a blue sphere and a green cone in my way when I run in simulator mode?

When you run in simulator mode, typically there will be VR Juggler simulator displays open showing the 3D space you are simulating. These are a special type of VR Juggler display. Instead of basing its viewpoint on the head position of one of the users, the viewpoint is controlled by a separate “camera” that is just another positional device. The Sim Display also draws certain objects to help visualize the environment. For example, the heads of users are represented as blue spheres with gray eyes, and a wand (if present) is drawn as a green cone. These are seen only in simulator displays and are there to help you if you are doing development using the simulator environment.

### 3.3.

How do I port a CAVElib™ application to VR Juggler?

Refer to the CAVElib-to-Juggler porting guide in the *Programmer's Guide* for details and code examples. For something quick, an overview follows.

In a VR Juggler `vjApp` class, there are five functions which you'll find immediately important: one for initialization, one for drawing (OpenGL only), and three for computation (I usually use only one of these). The two initialization functions are called upon execution of the application:

1. `init()` (called once when the application starts)
2. `contextInit()` (called once per window/graphics context creation)

The other three are called repeatedly in this order:

1. `preFrame()`
2. `draw()` (called once per open display)
3. `postFrame()`

These are the conceptual steps your application will take and should thus influence the structure of your application:

1. Initialize your data in `init()`.
2. Initialize your display contexts and texture objects in `contextInit()`. You will need to keep track of the ID/index per graphics window. The `vjContextData` class will help you with this.
3. Do your computations (such as modifying your data with VR Juggler input devices) in `preFrame()` and/or `postFrame()`.
4. Don't do your computations in `draw()`. Instead, just read your current data state, and draw (render) it. If you have display lists or texture objects, you will need to keep track of the ID/index per graphics window and call them accordingly.

Note that `draw()` is threaded, so may be called during `preFrame()` or during `postFrame()`. `contextInit()` is similar.

3.4.

I'm a CAVELib™ user that's trying to learn VR Juggler for a new project. Is there an easy way to setup a 2D projection on the walls? In CAVELib™ you'd do a `CAVEWallTransform()` which puts you in a 2D projection on the current wall so you can do 2D overlay stuff (disable lights & depth test, then a lot of `glVertex2f()`).

This is OpenGL specific. Turn off the depth buffer testing, and then draw your 3D geometry at the same position that the wall is at. Make sure you draw it last too. Also, don't transform the geometry with your navigation matrix.

3.5.

I am used to CAVELib™ capabilities. Is there a way to tell which wall you're drawing on?

You will know what wall you're drawing on by the position of your geometry. This way, you won't have to do any thing weird or morally wrong, like draw 2D in a 3D VR environment. Personally though, I wouldn't ever draw just on one wall, since technically the user isn't supposed to know the walls are there (that's what we call immersion). It could ruin the effect for them. I'd just put it up somewhere convenient in 3D space (if that happens to be the wall, then I wouldn't let that stop me. The depth buffer trick (described in the previous answer) will ensure that you'll always see it, but be careful since it can really confuse your users if something gets positioned in front of your head up display.

3.6.

How do I port a GLUT application to VR Juggler?

Please refer to the VR Juggler *Programmer's Guide* for a discussion on this topic and for example code.

3.7.

When I compile, the file `vector` (or `algorithm` or `iostream` etc.) cannot be found

VR Juggler depends on the C++ compiler implementing the latest C++ standard. This is especially true in the case of Standard Template Library support. The latest VR Juggler code also uses the standard `iostream` classes (part of the `std` namespace). If your compiler does not support these features, it cannot be used to compile VR Juggler or any VR Juggler applications. Make sure that you are using one of the supported compilers. If not, you will need to upgrade.

3.8.

When mouse/wand clicks are setup to register as digital button presses it is hard to get just one click

You will want to look at edge triggering on vjDigital devices.. An edge trigger gives you 4 states: on, off, just-on, or just-off. VR Juggler digital devices do edge triggering for you automatically. You will want to test for “just-on” so that your presses do not keep retriggering. For example, you can test it in the following to see when the button is first pressed:

```
if ( vjDigital::TOGGLE_ON == vjDigitalProxy::getData() )
{
    // Button just pressed
}
```

## VR Juggler Development

- 4.1. How can I submit changes/patches?

This question is too broad to be answered concisely here. Lucky for you, there is an entire chapter in the *Juggler Team Guide* just on exactly this topic.

- 4.2. How do I join the VR Juggler development team?

It's easy: just email us [mailto:vrjuggler-contact@vrjuggler.org]. Currently there is much to be done, especially in the device driver department. We are very interested in working with anyone, especially people willing to port their drivers to Juggler. Contact us directly [mailto:vrjuggler-contact@vrjuggler.org] with your thoughts, ideas, and be ready to work on a cool and upcoming player in the world of VR! (shameless plug)

- 4.3. I wrote and registered my device driver correctly, but VR Juggler cannot find it.

When the compiler sees the following code, it thinks that no one is using the code. As a result, it optimizes this code out when you link against your library.

```
#include Input/InputManager/vjDeviceFactory.h
vjDeviceConstructorMyButtonDevice* this_ptr_not_used =
    new vjDeviceConstructorMyButtonDevice;
```

The solution is to make sure that the compiler is told to include *all* library symbols. On IRIX with the MIPSpro Compilers, you will need to use the *-all* option when linking the application. With GCC, the equivalent options are *-Wl,-whole-archive* and *-Wl,-no-whole-archive*. Wrap these around statically listings of statically linked libraries. For example:

```
$(LINK) -o app --Wl,-Bstatic --Wl,-whole-archive -lJuggler \
    -lmy_driver_lib --Wl,-no-whole-archive --Wl,-Bdynamic
```

## Troubleshooting: General Compiling

- 5.1. When I compile a sample application, the compiler cannot find any VR Juggler headers.

The most common reason for this is that you did not set the environment variable `$VJ_BASE_DIR`. This is required for compiling as well as running because the makefiles for the shipped sample applications reference

`$VJ_BASE_DIR` when telling the compiler where to look for files. VR Juggler itself needs `$VJ_BASE_DIR` at run time in order to find the configuration description information.

- 5.2. When I compile a sample application, the compiler cannot find any VR Juggler libraries.

Refer to the previous answer.

- 5.3. I just compiled on Platform X, but now my application's makefile does not work at all on Platform Y.

In general, different platforms have different development environments. It is very difficult to guarantee that a compiler will behave exactly the same between two different platforms. This is especially true when moving from a Win32 environment to a version of UNIX or vice versa. The sample applications shipped with VR Juggler binary distributions all have their own makefile (unsurprisingly named `Makefile`) that was generated for use on the specific platform where VR Juggler was compiled. The fifth line of these files names this platform. This information is there to tell you where you can expect the makefile to work. If you have based your application's makefile on one of these samples, you should expect to do some work to get your application to build on another platform. While the code will not have to change (ideally), the way the compiling works probably will change. That is just a fact of life when working with multiple platforms. While it is theoretically possible to use the same single makefile on many different platforms, VR Juggler itself does not offer such a solution. That is most definitely not within the scope of the VR Juggler project.

## Troubleshooting: OpenGL Performer

- 6.1. Why does texturing not work? I copied the `pfNav` sample!

An old version of Juggler had a `pfDisable()` in its `pfNav` sample program. Remove this and the `pfOverride()` line, and you'll be golden.

- 6.2. Why don't my Performer-based VR Juggler applications compile on Linux?

Maybe you aren't using the `-O` option to your compiler. You must compile with the `-O` option when using Performer 2.3, 2.4, or 2.5 for LINUX [<http://www.sgi.com/software/performer/linux.html>]. See the Performer LINUX bug-list (item SCR 767188) [<http://www.sgi.com/software/performer/linux-buglist.html>] for more info.

- 6.3. Why don't my Performer-based VR Juggler applications compile on IRIX? I get an error that says "be.so not found".

Some SGI systems, specifically those using the MIPSpro Compiler version 7.3, need another directory added to the `$LD_LIBRARY_PATH` environment variable. It can be set as follows:

```
% setenv LD_LIBRARY_PATH $VJ_BASE_DIR/lib32:/usr/lib32/cmplrs
```

You will need to set this if your compiler reports it has trouble finding the file `be.so`. For more information on environment variables, refer to the *Getting Started Guide* [<http://www.vrjuggler.org/html/getting.started/>].